ECE 661: Homework 4

Kai Chi, CHAN

1 Description of Weighted RANSAC

Since there are usually a lot of noisy correspondences between two planar images, it is neceesary to select some of the correct correspondences to compute the homography. In this homework, weighted RANSAC is used to compute a homography.

Weighted RANSAC (referenced from HW4 specification)

1. Determine the noisy set of correspondences (100-1000) between two images using NCC matching of Harris corner points

2. Determine the number of trials N by

$$N = \frac{log(1-p)}{log(1-(1-\epsilon)^s)},$$

   where p is the probability of at least one of the random samples of s points is free from outliers and ϵ is the probability that any selected data point is an outlier.

3. For N trials

   (a) Select 4 different random correspondences from the complete set using weighted sampling (from HW4 specification)

   (b) Compute the homography defined by the 4 selected correspondence (from HW1)

   (c) Determince how many matches from the full set (inlier set) are consistent with this homography. The distance between the transformed point (x^t, y^t) and the corresponding point (x^c, y^c) is defined by $\sqrt{(x^t - x^c)^2 + (y^t - y^c)^2}$. If the distance is less than a threshold, the match is consistent with the homography.

   (d) Keep the homography with the largest number of inliers

4. Compute the final homography based on all the inlier data points using the normalized DLT algorithm for 2D homography which can be found in Algorithm 4.2 in the textbook. The computation of the similarity transformation T is shown below.

   A similarity transformation can be represented by the matrix

$$\begin{bmatrix} scos\theta & -ssin\theta & t_x \\ ssin\theta & scos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

   Since T only consists of a translation and scaling,

$$T = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

To take points $\vec{x_i} = \begin{bmatrix} x_i & y_i & 1 \end{bmatrix}^T$ to a new set of points $\vec{\tilde{x}_i} = \begin{bmatrix} \tilde{x_i} & \tilde{y_i} & 1 \end{bmatrix}^T$ such that the centroid of the points $\vec{\tilde{x}_i}$ is the coordinate origin $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$,

$$\begin{aligned} \overline{\vec{\tilde{x}_i}} &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ T\overline{\vec{x_i}} &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \overline{x_i} \\ \overline{y_i} \\ 1 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned}$$

$$t_x = -s\overline{x_i} \tag{1}$$

$$t_y = -s\overline{y_i} \tag{2}$$

Since the average distance of m points $\vec{\tilde{x}_i}$ from the origin is $\sqrt{2}$,

$$\begin{aligned} \frac{\sum_{i=1}^{m} \sqrt{\tilde{x_i}^2 + \tilde{y_i}^2}}{m} &= \sqrt{2} \\ \frac{\sum_{i=1}^{m} \sqrt{(sx_i + t_x)^2 + (sy_i + t_y)^2}}{m} &= \sqrt{2} \end{aligned}$$

From Equation (1) and (2),

$$\frac{\sum_{i=1}^{m} \sqrt{(sx_i - s\overline{x_i})^2 + (sy_i - s\overline{y_i})^2}}{m} = \sqrt{2}$$

$$s = \frac{m\sqrt{2}}{\sum_{i=1}^{m} \sqrt{(x_i - \overline{x_i})^2 + (y_i - \overline{y_i})^2}} \tag{3}$$

From Equation (1),(2) and (3), T can be computed. Similarly, the transformation T' can be computed.

2 Description of the mapping between the original image and the transformed image

Suppose X is a homogeneous coordinate of the original image and $X^{'}$ is the homogeneous coordinate of the transformed image. The mapping between the original image and the transformed image is

$$X^{'} = HX$$

The boundary of the transformed image is found by mapping the four corners of the original image to the transformed image. The smallest and largest x and y coordinates of the transformed image is the left upper and right lower coordinates of the boundary box containing the transformed image.

The transformed image is scaled so that the width of the boundary box equals to the width of the original image.

Since a point in the boundary box doesn't necessary to map to a pixel in the original image, bilinear interpolation is used.

Suppose f is a function mapping from an image coordinate to the intensity at the image coordinate.

Using the symbols in Figure 1, the linear interpolation in the x-direction is

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}),$$

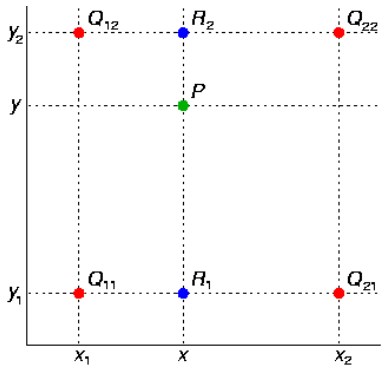


Figure 1: Bilinear interpolation

where $R_1 = (x, y_1)$,

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}),$$

where $R_2 = (x, y_2)$.

By interpolating in the y-direction,

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$

As a result, the intensity at (x, y) is

$$\begin{aligned} f(x,y) \quad \approx \quad & \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y) \\ & + \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1) \end{aligned}$$

3 Parameter Setting

In this homework, the parameters are tuned so that the number of correspondence is large (100-1000). The parameters are summarized in the following table. The values of the parameters for each image is shown in Table 2.

Table 1: Parameter description of Harris corner detection

Parameter	Description
M_S	Window size of Sobel operator
M_H	Window size of Harris corner detector
th_R	Threshold of the corner response
W_R	Window size of the local maximum of the corner response
m_{ncc}	Window size of the feature descriptor using NCC
th_{ncc}	Threshold of the smallest NCC value considered to be a corner
R_{ncc}	Ratio of $\frac{secondlargestNCC}{firstlargestNCC}$

In the Weigthed RANSAC, after applying the homography computed by the 4 randomly selected correspondences, the transformed point and its corresponding point are matched if their distance is less than 15.

Table 2: Parameter setting of Harris corner detection

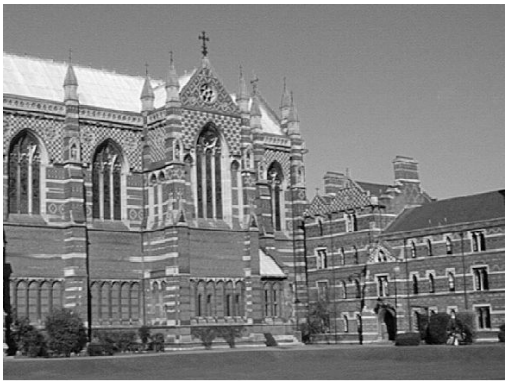
	M_S	M_H	th_R	W_R	m_{ncc}	th_{ncc}	R_{ncc}
sample_a.jpg	3	5	4000000	5	13	0.5	0.85
sample_b.jpg	3	5	3000000	5	13	0.5	0.85
booklet1.jpg	3	5	4000000	5	13	0.5	0.95
booklet2.jpg	3	5	3000000	5	13	0.5	0.95
box1.jpg	3	5	4000000	5	13	0.5	0.95
box2.jpg	3	5	3000000	5	13	0.5	0.95

4 Results

Table 3: Result summary of the Weighted RANSAC

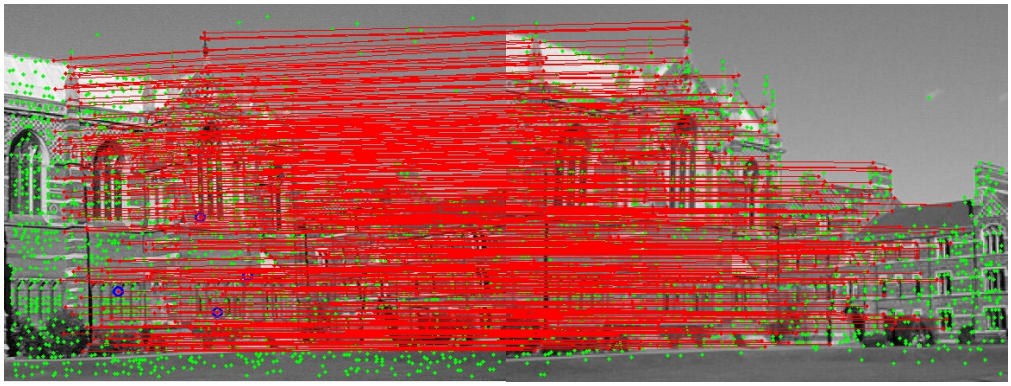
	# of features (left image, right image)	# of correspondences	P(outlier), ϵ	# of trials	# of inliers
First pair of images	2046, 1826	653	0.377	28	430
Second pair of images	1582, 1826	723	0.582	148	302
Third pair of images	1763, 1662	605	0.597	171	244

In the figures below, green dots represent image features. Red lines represent feature correspondences. Blue circles in the left images represent the features of the corresponding 4 selected correspondences with the largest number of inliers.



(a) sample-a.jpg

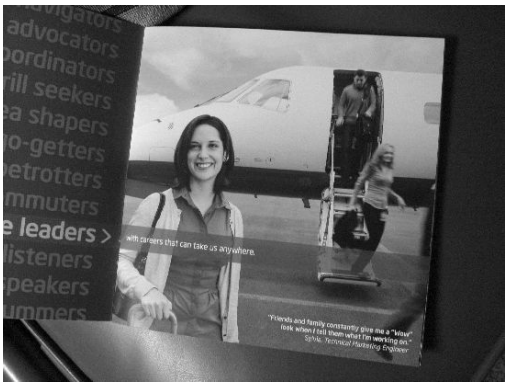
(b) sample-b.jpg



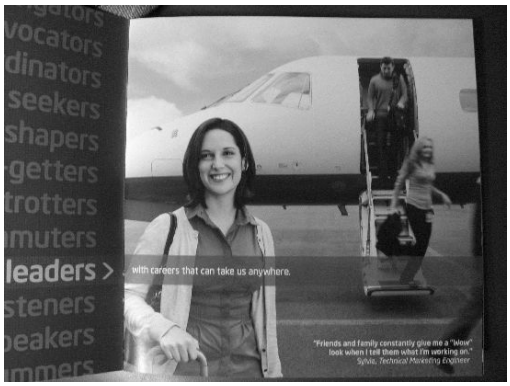
(c) Inlier Matches

(d) Transformation of the first input to look like the second

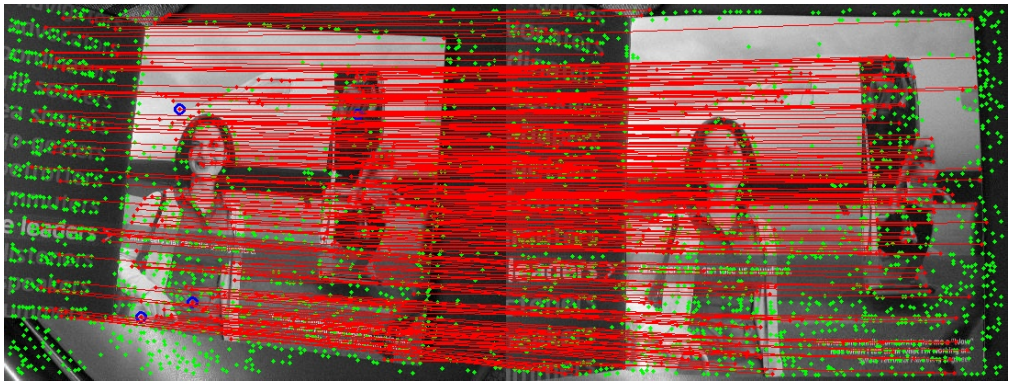
Figure 2: First pair of images and results



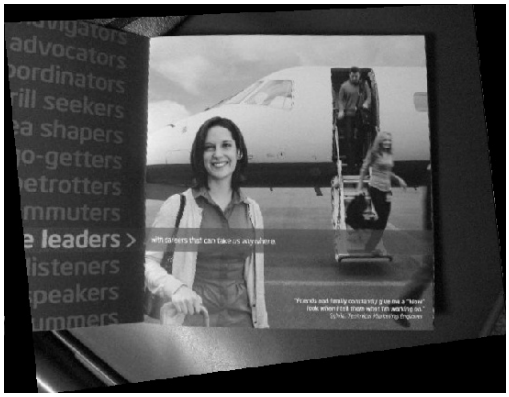
(a) booklet1.jpg



(b) booklet2.jpg

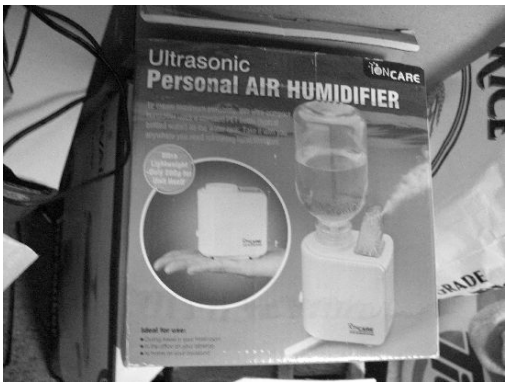


(c) Inlier Matches

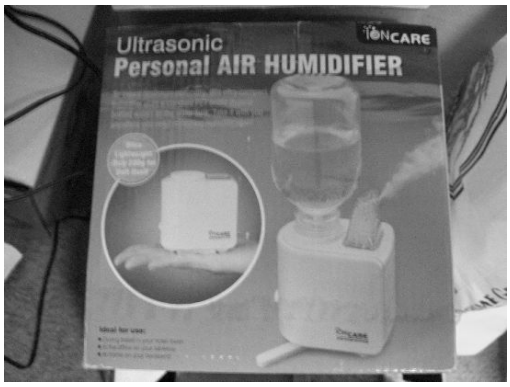


(d) Transformation of the first input to look like the second

Figure 3: Second pair of images and results

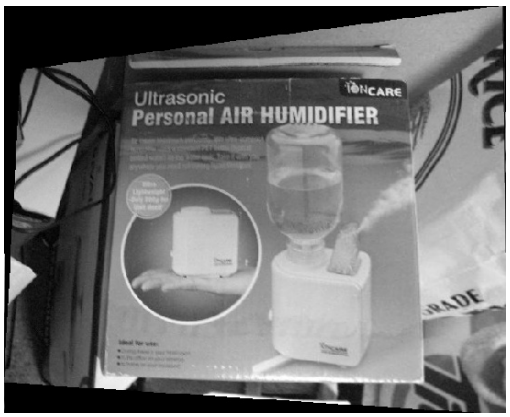


(a) box1.jpg



(b) box2.jpg

(c) Inlier Matches



(d) Transformation of the first input to look like the second

Figure 4: Third pair of images and results

5 Source code

```
1
2  #include "stdafx.h"
3  #include <cv.h>
4  #include <highgui.h>
5  #include <stdlib.h>
6  #include <stdio.h>
7  #include <time.h>
8  #include <limits.h>
9
10 #define MASK_SIZE 5
11 #define SIZE 13
12
13 typedef struct _struct_feature{
14   double pt[2];
15   CvMat* neighborhood;
16   _struct_feature *match;
17   double score;
18   _struct_feature *next;
19 }FEATURE;
20
21 FEATURE* init_feature(double* pt, CvMat* neighborhood)
22 {
23   FEATURE* f;
24   f = (FEATURE*) malloc(sizeof(FEATURE));
25   if (f==NULL)
26   {
27     printf("Error in creating feature...\n");
28     exit(1);
29   }
30   f->pt[0] = pt[0];
31   f->pt[1] = pt[1];
32   f->neighborhood = neighborhood;
33   f->match = NULL;
34   f->score = -999;
35   f->next = NULL;
36   return f;
37 }
38
39 int feature_no(FEATURE *f)
40 {
41   int length = 0;
42   FEATURE* cur = f;
43   while (cur!=NULL)
44   {
45     length++;
46     cur = cur->next;
47   }
48   return length;
49 }
50
51 FEATURE* weightedSamplingAlgorithm(FEATURE* f)
52 {
53   double sumOfWeight=0;
54   FEATURE* cur;
55   cur = f->next;
56   while (cur!=NULL)
57   {
58     if (cur->match!=NULL)
59     {
60       sumOfWeight+=cur->score;
61     }
62     cur = cur->next;
63   }
64
65   while (1)
66   {
67     double u = (double) rand() / (double) RAND_MAX;
```

```
68
69      double c = 0;
70
71      cur = f->next;
72
73      while (cur!=NULL)
74      {
75        if (cur->match!=NULL)
76        {
77          c = c+cur->score;
78
79          if (c>= (u*sumOfWeight))
80          {
81            return cur;
82          }
83        }
84        cur = cur->next;
85      }
86    }
87  }
88
89  void PrintMatrix(CvMat *Matrix,char *name)
90  {
91    printf("%s\n", name);
92      for(int i=0;i<Matrix->rows;i++)
93      {
94          for(int j=0;j<Matrix->cols;j++)
95          {
96              printf("%.3f\t",cvGet2D(Matrix,i,j).val[0]);
97          }
98          printf("\n");
99      }
100   printf("\n");
101 }
102
103 void Sobel(IplImage* img, CvMat* dx, CvMat* dy)
104 {
105
106   int mask_dx[3][3] = {-1, 0, 1, -2, 0, 2, -1, 0, 1};
107   int mask_dy[3][3] = {1, 2, 1, 0, 0, 0, -1, -2, -1};
108
109   int sum_dx = 0;
110   int sum_dy = 0;
111
112   int width = img->width;
113   int height = img->height;
114   int step = img->widthStep;
115   uchar *data = (uchar *)img->imageData;
116
117   cvZero(dx);
118   cvZero(dy);
119
120   for (int x=1; x<width-1; x++)
121   {
122     for (int y=1; y<height-1; y++)
123     {
124       sum_dx = 0;
125       sum_dy = 0;
126
127       for (int mask_x = 0; mask_x < 3; mask_x++)
128       {
129         for (int mask_y = 0; mask_y < 3; mask_y++)
130         {
131           sum_dx += data[(y+mask_y-1)*step+(x+mask_x-1)]*mask_dx[mask_y][mask_x];
132           sum_dy += data[(y+mask_y-1)*step+(x+mask_x-1)]*mask_dy[mask_y][mask_x];
133         }
134       }
135
136       cvmSet(dx, y, x, (double) sum_dx);
137       cvmSet(dy, y, x, (double) sum_dy);
```

```
138       }
139     }
140
141     return;
142   }
143
144   CvMat* HarrisCornerDetector(IplImage* img)
145   {
146     int height,width,step,channels,depth;
147     uchar *data, *data_x, *data_y;
148
149     // get the image data
150     height    = img->height;
151     width     = img->width;
152     depth     = img->depth;
153     step      = img->widthStep;
154     channels  = img->nChannels;
155     data      = (uchar *)img->imageData;
156
157     CvMat* img_x = cvCreateMat(height, width, CV_64FC1);
158     CvMat* img_y = cvCreateMat(height, width, CV_64FC1);
159
160     Sobel(img, img_x, img_y);
161
162     CvMat* w = cvCreateMat(MASK_SIZE, MASK_SIZE, CV_64FC1);
163     CvMat* h = cvCreateMat(img->height, img->width, CV_64FC3);
164     CvMat* M = cvCreateMat(2, 2, CV_64FC1);
165     CvMat* M_U = cvCreateMat(2, 2, CV_64FC1);
166     CvMat* M_D = cvCreateMat(2, 2, CV_64FC1);
167     CvMat* M_V = cvCreateMat(2, 2, CV_64FC1);
168     CvMat* r = cvCreateMat(img->height, img->width, CV_64FC1);
169     CvMat* r1 = cvCreateMat(img->height, img->width, CV_64FC1);
170
171     cvZero(h);
172     cvZero(r);
173
174     for (int i=0; i<w->rows; i++)
175       for (int j=0; j<w->cols; j++)
176         cvmSet(w, i, j, 1.0/MASK_SIZE);
177
178     for (int i=MASK_SIZE/2; i<img->height-MASK_SIZE/2; i++)
179     {
180       for (int j=MASK_SIZE/2; j<img->width-MASK_SIZE/2; j++)
181       {
182         CvScalar s=cvGet2D(h,i,j);
183         for (int w_x=-MASK_SIZE/2; w_x<MASK_SIZE/2+1; w_x++)
184         {
185           for (int w_y=-MASK_SIZE/2; w_y<MASK_SIZE/2+1; w_y++)
186           {
187             s.val[0]+=cvmGet(w, w_y+MASK_SIZE/2, w_x+MASK_SIZE/2)*cvmGet(img_x, (i+w_y), (j+
                  w_x))*cvmGet(img_x, (i+w_y), (j+w_x));//data_x[(i+w_y)*step+(j+w_x)];
188             s.val[1]+=cvmGet(w, w_y+MASK_SIZE/2, w_x+MASK_SIZE/2)*cvmGet(img_x, (i+w_y), (j+
                  w_x))*cvmGet(img_y, (i+w_y), (j+w_x));//data_y[(i+w_y)*step+(j+w_x)];
189             s.val[2]+=cvmGet(w, w_y+MASK_SIZE/2, w_x+MASK_SIZE/2)*cvmGet(img_y, (i+w_y), (j+
                  w_x))*cvmGet(img_y, (i+w_y), (j+w_x));//data_y[(i+w_y)*step+(j+w_x)];
190           }
191         }
192         cvSet2D(h, i, j, s);
193         cvmSet(M, 0, 0, s.val[0]);
194         cvmSet(M, 0, 1, s.val[1]);
195         cvmSet(M, 1, 0, s.val[1]);
196         cvmSet(M, 1, 1, s.val[2]);
197         cvSVD(M, M_D, M_U, M_V);
198
199         double lambda1 = cvmGet(M_D, 0, 0);
200         double lambda2 = cvmGet(M_D, 1, 1);
201
202         cvmSet(r, i, j, lambda1*lambda2 - 0.04*(lambda1+lambda2)*(lambda1+lambda2));
203       }
204     }
```

```
205
206     cvCopy(r,r1);
207
208     for (int i=MASK_SIZE/2; i<img->height-MASK_SIZE/2; i++)
209     {
210       for (int j=MASK_SIZE/2; j<img->width-MASK_SIZE/2; j++)
211       {
212         for (int w_x=-MASK_SIZE/2; w_x<MASK_SIZE/2+1; w_x++)
213         {
214           for (int w_y=-MASK_SIZE/2; w_y<MASK_SIZE/2+1; w_y++)
215           {
216             if (cvmGet(r, i, j)<cvmGet(r, i+w_y, j+w_x))
217             {
218               cvmSet(r1, i, j, 0);
219             }
220           }
221         }
222       }
223     }
224
225     return r1;
226   }
227
228   CvMat* neighbour(IplImage *img, CvPoint x, int size)
229   {
230     CvMat* m = cvCreateMat(size, size, CV_64FC1);
231
232     int step;
233     uchar *data;
234
235     // get the image data
236     step      = img->widthStep;
237     data      = (uchar *)img->imageData;
238
239     for (int i=-size/2; i<=size/2; i++)
240     {
241       for (int j=-size/2; j<=size/2; j++)
242       {
243         cvmSet(m, i+size/2, j+size/2, data[(i+x.y)*step+(j+x.x)]);
244       }
245     }
246
247     return m;
248   }
249
250   FEATURE* getFeature(IplImage* img,CvMat* r, long long int thres)
251   {
252     FEATURE *cur, *prev, *head;
253     head = (FEATURE*) malloc(sizeof(FEATURE));;
254     cur = head;
255     for (int i=SIZE/2; i<img->height-SIZE/2; i++)
256     {
257       for (int j=SIZE/2; j<img->width-SIZE/2; j++)
258       {
259         if (cvmGet(r, i, j)>thres)
260         {
261           prev = cur;
262           double pt[2];
263           pt[0] = (double)j;
264           pt[1] = (double)i;
265           cur = init_feature(pt, neighbour(img, cvPoint(j, i), SIZE));
266           prev->next = cur;
267         }
268       }
269     }
270     return head;
271   }
272
273   double ncc(CvMat* I1, CvMat* I2)
274   {
```

```
275    double I1_mean = cvMean(I1);
276    double I2_mean = cvMean(I2);
277
278    CvMat* I1_mean_matrix = cvCreateMat(SIZE, SIZE, CV_64FC1);
279    CvMat* I2_mean_matrix = cvCreateMat(SIZE, SIZE, CV_64FC1);
280    CvMat* I12 = cvCreateMat(SIZE, SIZE, CV_64FC1);
281    CvMat* I1_sq = cvCreateMat(SIZE, SIZE, CV_64FC1);
282    CvMat* I2_sq = cvCreateMat(SIZE, SIZE, CV_64FC1);
283
284    cvAddS(I1, cvScalar(-I1_mean), I1_mean_matrix);
285    cvAddS(I2, cvScalar(-I2_mean), I2_mean_matrix);
286
287    cvMul(I1_mean_matrix, I2_mean_matrix, I12);
288
289    cvMul(I1_mean_matrix, I1_mean_matrix, I1_sq);
290    cvMul(I2_mean_matrix, I2_mean_matrix, I2_sq);
291
292    double value = cvSum(I12).val[0]/sqrt(cvSum(I1_sq).val[0]*cvSum(I2_sq).val[0]);
293
294    cvReleaseMat(&I1_mean_matrix);
295    cvReleaseMat(&I2_mean_matrix);
296    cvReleaseMat(&I12);
297    cvReleaseMat(&I1_sq);
298    cvReleaseMat(&I2_sq);
299
300    return value;
301  }
302
303  void feature_matching_NCC(FEATURE* f1, FEATURE* f2, double thres, double ratio)
304  {
305    double score, score2;
306    int numberOfFeature = feature_no(f1);
307    FEATURE *cur1, *cur2;
308
309    cur1 = f1;
310
311    while (cur1->next!=NULL)
312    {
313      cur1 = cur1->next;
314      score = -100;
315      score2 = -100;
316      cur2 = f2;
317
318      while (cur2->next!=NULL)
319      {
320        cur2 = cur2->next;
321        double value = ncc(cur1->neighborhood, cur2->neighborhood);
322
323        if (score<value && value >thres)
324        {
325          score2 = score;
326          score = value;
327
328          cur1->match = cur2;
329          cur1->score = score;
330
331          cur2->match = cur1;
332          cur2->score = score;
333        }
334      }
335
336      if (score>0 && score2>0)
337      {
338        if (score2/score>ratio)
339        {
340          cur1->match->match = NULL;
341          cur1->match->score = -999;
342
343          cur1->match = NULL;
344          cur1->score = -999;
```

```
345         }
346       }
347     }
348   }
349
350   void markFeatures(IplImage *img, FEATURE * f)
351   {
352     FEATURE *cur;
353     cur = f->next;
354     while (cur!=NULL)
355     {
356       cvCircle(img, cvPoint((int)cur->pt[0], (int)cur->pt[1]), 0, cvScalar(0, 255, 0), 5);
357       cur = cur->next;
358     }
359   }
360
361   void drawMatching(IplImage *img_h, FEATURE* f)
362   {
363     FEATURE* cur;
364     cur = f->next;
365     int i=0;
366     while (cur!=NULL)
367     {
368       if (cur->match!=NULL)
369       {
370         cvCircle(img_h,cvPoint((int)cur->pt[0], (int)cur->pt[1]), 0, cvScalar(0,0,255),5);
371         cvCircle(img_h,cvPoint((int)(cur->match->pt[0]+img_h->width*0.5), cur->match->pt[1]),
              0, cvScalar(0,0,255),5);
372         cvLine(img_h, cvPoint((int)cur->pt[0], (int) cur->pt[1]), cvPoint((int)(cur->match->pt
              [0]+img_h->width*0.5), (int) cur->match->pt[1]), cvScalar(0,0,255));
373         i++;
374       }
375       cur = cur->next;
376     }
377     return;
378   }
379
380   FEATURE* computeCorrespondences(FEATURE* f, FEATURE** c)
381   {
382     for (int i=0; i<4; i++)
383     {
384       c[i] = weightedSamplingAlgorithm(f);
385       for (int j=i-1; j>=0; j--)
386       {
387         if (c[i] == c[j])
388         {
389           i--;
390           break;
391         }
392       }
393     }
394
395     FEATURE* corr_head, *cur;
396     corr_head = (FEATURE*) malloc(sizeof(FEATURE));
397     cur = corr_head;
398
399     for (int i=0; i<4; i++)
400     {
401       cur->next = init_feature(c[i]->pt, NULL);
402       cur->next->match = init_feature(c[i]->match->pt, NULL);
403       cur = cur->next;
404     }
405
406     return corr_head;
407   }
408
409   double* transform_pt(CvMat* H, double* pt)
410   {
411     double* t_pt;
412     t_pt = (double*) malloc(2*sizeof(double));
```

```
413   CvMat *im_coord=cvCreateMat(3,1,CV_64FC1);
414   CvMat *t_im_coord=cvCreateMat(3,1,CV_64FC1);
415
416   cvmSet(im_coord, 0, 0, pt[0]);
417   cvmSet(im_coord, 1, 0, pt[1]);
418   cvmSet(im_coord, 2, 0, 1.0);
419
420   cvMatMul(H, im_coord, t_im_coord);
421
422   t_pt[0] = cvmGet(t_im_coord, 0, 0)/cvmGet(t_im_coord, 2, 0);
423   t_pt[1] = cvmGet(t_im_coord, 1, 0)/cvmGet(t_im_coord, 2, 0);
424
425   cvReleaseMat(&im_coord);
426   cvReleaseMat(&t_im_coord);
427
428   return t_pt;
429 }
430
431
432 FEATURE* computeInlierSet(FEATURE* f1_head, CvMat *H, double thres)
433 {
434
435   FEATURE* cur;
436   cur = f1_head->next;
437
438   FEATURE *cur_inlier, *inlier_head;
439
440   inlier_head = (FEATURE*) malloc(sizeof(FEATURE));
441   cur_inlier = inlier_head;
442
443   double *t_pt;
444
445   double diff;
446   while (cur!=NULL)
447   {
448     if (cur->match!=NULL)
449     {
450       t_pt = transform_pt(H, cur->pt);
451       diff = sqrt((double)((t_pt[0] - cur->match->pt[0])*(t_pt[0] - cur->match->pt[0]) + (
              t_pt[1] - cur->match->pt[1])*(t_pt[1] - cur->match->pt[1])));
452       if (diff<thres)
453       {
454         cur_inlier->next = init_feature(cur->pt, NULL);
455         cur_inlier->next->match = init_feature(cur->match->pt, NULL);
456         cur_inlier = cur_inlier->next;
457       }
458     }
459     cur = cur->next;
460   }
461
462   return inlier_head;
463 }
464
465
466 int computeInlierSetNumber(FEATURE* f1_head, CvMat *H, double thres, int* total)
467 {
468   FEATURE* cur;
469   cur = f1_head->next;
470
471   *total = 0;
472
473   double* t_pt;
474   int count = 0;
475   double diff;
476   while (cur!=NULL)
477   {
478     if (cur->match!=NULL)
479     {
480       *total = *total + 1;
481       t_pt = transform_pt(H, cur->pt);
```

```
482
483       diff = sqrt((t_pt[0] - cur->match->pt[0])*(t_pt[0] - cur->match->pt[0]) + (t_pt[1] -
              cur->match->pt[1])*(t_pt[1] - cur->match->pt[1]));
484
485       if (diff<thres)
486       {
487         count++;
488       }
489     }
490     cur = cur->next;
491   }
492
493   return count;
494 }
495
496 CvMat* computeHomographyInlier(FEATURE* inlier_head)
497 {
498   FEATURE* cur;
499   cur = inlier_head->next;
500   CvMat *H = cvCreateMat(3,3,CV_64FC1);
501
502   int noOfInlier = feature_no(cur);
503
504   double** im1_coord;
505   double** im2_coord;
506
507   im1_coord = (double**)malloc(noOfInlier*sizeof(*im1_coord));
508   im2_coord = (double**)malloc(noOfInlier*sizeof(*im2_coord));
509
510   for (int i = 0; i< noOfInlier; i++)
511   {
512     im1_coord[i] =(double*) malloc(2*sizeof(double));
513     im1_coord[i][0]=cur->pt[0];
514     im1_coord[i][1]=cur->pt[1];
515
516     im2_coord[i] =(double*) malloc(2*sizeof(double));
517     im2_coord[i][0]=cur->match->pt[0];
518     im2_coord[i][1]=cur->match->pt[1];
519
520     cur = cur->next;
521   }
522
523   if (cur!=NULL) printf("Error in calculating H...\n");
524
525   CvMat *matrix_coordinate = cvCreateMat(2*noOfInlier,9,CV_64FC1);
526   CvMat *matrix_W=cvCreateMat(2*noOfInlier,9,CV_64FC1);
527     CvMat *matrix_V=cvCreateMat(9,9,CV_64FC1);
528     CvMat *matrix_U=cvCreateMat(2*noOfInlier,2*noOfInlier,CV_64FC1);
529
530   for (int i=0; i<noOfInlier; i++)
531   {
532     cvmSet(matrix_coordinate,i*2,0,im1_coord[i][0]);
533     cvmSet(matrix_coordinate,i*2,1,im1_coord[i][1]);
534     cvmSet(matrix_coordinate,i*2,2,1.0);
535     cvmSet(matrix_coordinate,i*2,3,0);
536     cvmSet(matrix_coordinate,i*2,4,0);
537     cvmSet(matrix_coordinate,i*2,5,0);
538     cvmSet(matrix_coordinate,i*2,6,-im1_coord[i][0]*im2_coord[i][0]);
539     cvmSet(matrix_coordinate,i*2,7,-im1_coord[i][1]*im2_coord[i][0]);
540     cvmSet(matrix_coordinate,i*2,8,-im2_coord[i][0]);
541
542     cvmSet(matrix_coordinate,i*2+1,0,0);
543     cvmSet(matrix_coordinate,i*2+1,1,0);
544     cvmSet(matrix_coordinate,i*2+1,2,0);
545     cvmSet(matrix_coordinate,i*2+1,3,-im1_coord[i][0]);
546     cvmSet(matrix_coordinate,i*2+1,4,-im1_coord[i][1]);
547     cvmSet(matrix_coordinate,i*2+1,5,-1.0);
548     cvmSet(matrix_coordinate,i*2+1,6,im1_coord[i][0]*im2_coord[i][1]);
549     cvmSet(matrix_coordinate,i*2+1,7,im1_coord[i][1]*im2_coord[i][1]);
550     cvmSet(matrix_coordinate,i*2+1,8,im2_coord[i][1]);
```

```
551     }
552
553     cvSVD(matrix_coordinate,matrix_W,matrix_U,matrix_V);
554
555     cvmSet(H,0,0,cvmGet(matrix_V,0,8));
556     cvmSet(H,0,1,cvmGet(matrix_V,1,8));
557     cvmSet(H,0,2,cvmGet(matrix_V,2,8));
558     cvmSet(H,1,0,cvmGet(matrix_V,3,8));
559     cvmSet(H,1,1,cvmGet(matrix_V,4,8));
560     cvmSet(H,1,2,cvmGet(matrix_V,5,8));
561     cvmSet(H,2,0,cvmGet(matrix_V,6,8));
562     cvmSet(H,2,1,cvmGet(matrix_V,7,8));
563     cvmSet(H,2,2,cvmGet(matrix_V,8,8));
564
565     cvReleaseMat(&matrix_coordinate);
566     cvReleaseMat(&matrix_W);
567     cvReleaseMat(&matrix_U);
568     cvReleaseMat(&matrix_V);
569     for (int i = 0; i< noOfInlier; i++)
570     {
571       free(im1_coord[i]);
572       free(im2_coord[i]);
573     }
574     free(im1_coord);
575     free(im2_coord);
576
577     return H;
578   }
579
580   void computeT_Tpi(CvMat* T, CvMat*T_pi, FEATURE* inlier_head)
581   {
582     int N = feature_no(inlier_head->next);
583
584     if (N==0)
585       printf("Error: no of inlier = 0\n");
586
587     double x_sum, y_sum, x_mean, y_mean;
588     double x_pi_sum, y_pi_sum, x_pi_mean, y_pi_mean;
589
590     FEATURE* cur;
591     cur = inlier_head->next;
592
593     x_sum = 0.0;
594     y_sum = 0.0;
595     x_pi_sum = 0.0;
596     y_pi_sum = 0.0;
597
598     while(cur!=NULL)
599     {
600       x_sum += cur->pt[0];
601       y_sum += cur->pt[1];
602       x_pi_sum += cur->match->pt[0];
603       y_pi_sum += cur->match->pt[1];
604
605       cur = cur->next;
606     }
607
608     x_mean = x_sum/(double)N;
609     y_mean = y_sum/(double)N;
610     x_pi_mean = x_pi_sum/(double)N;
611     y_pi_mean = y_pi_sum/(double)N;
612
613     double ssd = 0.0;
614     double ssd_pi = 0.0;
615
616     cur = inlier_head->next;
617
618     while(cur!=NULL)
619     {
620       ssd += sqrt(pow(cur->pt[0]-x_mean,2) + pow(cur->pt[1]-y_mean,2));
```

```
621       ssd_pi += sqrt(pow(cur->match->pt[0]-x_pi_mean,2) + pow(cur->match->pt[1]-y_pi_mean,2));
622       cur = cur->next;
623     }
624
625     double s, s_pi;
626
627     if (ssd == 0 || ssd_pi ==0)
628     {
629       printf("error in computing T Tpi\n");
630       printf("x_mean = %.3lf y_mean = %.3lf x_pi_mean = %.3lf y_pi_mean = %.3lf\n",x_mean,
              y_mean, x_pi_mean, y_pi_mean);
631     }
632
633     s = (((double)N)*sqrt(2.0))/ssd;
634     s_pi = (((double)N)*sqrt(2.0))/ssd_pi;
635
636     double tx, ty, tx_pi, ty_pi;
637
638     tx = -s*x_mean;
639     ty = -s*y_mean;
640     tx_pi = -s_pi*x_pi_mean;
641     ty_pi = -s_pi*y_pi_mean;
642
643     cvZero(T);
644     cvmSet(T, 0, 0, s);
645     cvmSet(T, 0, 2, tx);
646     cvmSet(T, 1, 1, s);
647     cvmSet(T, 1, 2, ty);
648     cvmSet(T, 2, 2, 1);
649
650     cvZero(T_pi);
651     cvmSet(T_pi, 0, 0, s_pi);
652     cvmSet(T_pi, 0, 2, tx_pi);
653     cvmSet(T_pi, 1, 1, s_pi);
654     cvmSet(T_pi, 1, 2, ty_pi);
655     cvmSet(T_pi, 2, 2, 1);
656
657     return;
658   }
659
660   FEATURE* normalization(FEATURE* inlier_head, CvMat* T, CvMat* T_pi)
661   {
662     FEATURE* cur;
663     cur = inlier_head->next;
664
665     FEATURE *cur_n_inlier, *n_inlier_head;
666
667     n_inlier_head = (FEATURE*) malloc(sizeof(FEATURE));
668     cur_n_inlier = n_inlier_head;
669
670     double *t_pt, *t_pi_pt;
671
672     while (cur!=NULL)
673     {
674
675       t_pt = transform_pt(T, cur->pt);
676       t_pi_pt = transform_pt(T_pi, cur->match->pt);
677
678       double temp[2];
679       double temp_pi[2];
680
681       temp[0] = t_pt[0];
682       temp[1] = t_pt[1];
683       temp_pi[0] = t_pi_pt[0];
684       temp_pi[1] = t_pi_pt[1];
685
686       cur_n_inlier->next = init_feature(temp, NULL);
687       cur_n_inlier->next->match = init_feature(temp_pi, NULL);
688
689       cur_n_inlier = cur_n_inlier->next;
```

```
690       cur = cur->next;
691     }
692
693     return n_inlier_head;
694   }
695
696
697   CvMat* deNormalization(CvMat* H_DLT, CvMat* T, CvMat* T_pi)
698   {
699     CvMat* T_pi_inverse = cvCreateMat(3, 3, CV_64FC1);
700     cvInvert(T_pi, T_pi_inverse);
701     CvMat* T_pi_i_H = cvCreateMat(3, 3, CV_64FC1);
702     CvMat* T_pi_i_H_T = cvCreateMat(3, 3, CV_64FC1);
703
704     cvMatMul(T_pi_inverse, H_DLT, T_pi_i_H);
705     cvMatMul(T_pi_i_H, T, T_pi_i_H_T);
706
707     cvReleaseMat(&T_pi_inverse);
708     cvReleaseMat(&T_pi_i_H);
709
710     return T_pi_i_H_T;
711   }
712
713   CvMat* computeNormalizedDLT(FEATURE* inlier_head)
714   {
715     FEATURE* n_inlier_head;
716     CvMat* H_DLT;
717     CvMat* Denorm_H_DLT;
718     CvMat* T = cvCreateMat(3,3,CV_64FC1);
719     CvMat* T_pi = cvCreateMat(3,3,CV_64FC1);
720
721     computeT_Tpi(T, T_pi, inlier_head);
722
723     n_inlier_head = normalization(inlier_head, T, T_pi);
724     H_DLT = computeHomographyInlier(n_inlier_head);
725     Denorm_H_DLT = deNormalization(H_DLT, T, T_pi);
726
727     return Denorm_H_DLT;
728   }
729
730   void transform(IplImage *img, IplImage *dst, CvMat* matrix_homography, double scalingFactor,
          int min_x, int min_y)
731   {
732     cvZero(dst);
733
734     int channels_s, channels_d, imgstep_s, imgstep_d,x_w,y_w;
735       unsigned char *imgData_d, *imgData_s;
736
737     channels_d = dst->nChannels;
738       imgstep_d  = dst->widthStep / sizeof (unsigned char); // Values per row
739     channels_s = img->nChannels;
740       imgstep_s  = img->widthStep / sizeof (unsigned char); // Values per row
741       imgData_d  = (unsigned char *)dst->imageData;
742     imgData_s = (unsigned char *)img->imageData;
743
744     CvMat *m_image = cvCreateMat( 3, 1, CV_64FC1 );
745     CvMat *m_world = cvCreateMat( 3, 1, CV_64FC1 );
746
747     cvmSet(m_world,2,0,1);
748
749     double x_f, y_f;
750     int x1, x2, y1, y2;
751
752     for (int i = 0; i < dst->height; i++)
753     {
754       for (int j=0; j<dst->width;j++)
755       {
756         cvmSet(m_world,0,0,(int)(j*scalingFactor+min_x));
757         cvmSet(m_world,1,0,(int)(i*scalingFactor+min_y));
758
```

```
759         cvMatMul(matrix_homography,m_world,m_image);
760
761         x_f=cvmGet(m_image,0,0)/cvmGet(m_image,2,0);
762         y_f=cvmGet(m_image,1,0)/cvmGet(m_image,2,0);
763
764         x1= (int) floor(x_f);
765         y1= (int) floor(y_f);
766         x2= (int) ceil(x_f);
767         y2= (int) ceil(y_f);
768
769         x_w=j;
770         y_w=i;
771         if (x1<img->width&&x1>=0&&x2<img->width&&x2>=0&&y1<img->height&&y1>=0&&y2<img->height
              &&y2>=0)
772         {
773           if (x1==x2||y1==y2)
774           {
775             if (x1==x2&&y1!=y2)
776             {
777               imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 0] =
778                 imgData_s[(y1 * imgstep_s) + (x1 * channels_s) + 0]*(y2-y_f)/(y2-y1)
779                 +imgData_s[(y2 * imgstep_s) + (x1 * channels_s) + 0]*(y_f-y1)/(y2-y1);
780               imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 1] = imgData_s[(y1 *
                    imgstep_s) + (x1 * channels_s) + 1]*(y2-y_f)/(y2-y1)
781                                           +imgData_s[(y2 * imgstep_s) + (x1 * channels_s) +
                                                1]*(y_f-y1)/(y2-y1);
782               imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 2] = imgData_s[(y1 *
                    imgstep_s) + (x1 * channels_s) + 2]*(y2-y_f)/(y2-y1)
783                                           +imgData_s[(y2 * imgstep_s) + (x1 * channels_s) +
                                                2]*(y_f-y1)/(y2-y1);
784             }
785             else if (x1!=x2&&y1==y2)
786             {
787               imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 0] = imgData_s[(y1 *
                    imgstep_s) + (x1 * channels_s) + 0]*(x2-x_f)/(x2-x1)
788                                           +imgData_s[(y1 * imgstep_s) + (x2 * channels_s) +
                                                0]*(x_f-x1)/(x2-x1);
789               imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 1] = imgData_s[(y1 *
                    imgstep_s) + (x1 * channels_s) + 1]*(x2-x_f)/(x2-x1)
790                                           +imgData_s[(y1 * imgstep_s) + (x2 * channels_s) +
                                                1]*(x_f-x1)/(x2-x1);
791               imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 2] = imgData_s[(y1 *
                    imgstep_s) + (x1 * channels_s) + 2]*(x2-x_f)/(x2-x1)
792                                           +imgData_s[(y1 * imgstep_s) + (x2 * channels_s) +
                                                2]*(x_f-x1)/(x2-x1);
793             }
794             else
795             {
796               imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 0] = imgData_s[(y1 *
                    imgstep_s) + (x1 * channels_s) + 0];
797               imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 1] = imgData_s[(y1 *
                    imgstep_s) + (x1 * channels_s) + 1];
798               imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 2] = imgData_s[(y1 *
                    imgstep_s) + (x1 * channels_s) + 2];
799             }
800           }
801           else
802           {
803             imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 0] = imgData_s[(y1 * imgstep_s)
                  + (x1 * channels_s) + 0]*(x2-x_f)*(y2-y_f)/((x2-x1)*(y2-y1))
804                                         +imgData_s[(y1 * imgstep_s) + (x2 * channels_s) + 0]*(
                                              x_f-x1)*(y2-y_f)/((x2-x1)*(y2-y1))
805                                         +imgData_s[(y2 * imgstep_s) + (x1 * channels_s) + 0]*(
                                              x2-x_f)*(y_f-y1)/((x2-x1)*(y2-y1))
806                                         +imgData_s[(y2 * imgstep_s) + (x2 * channels_s) + 0]*(
                                              x_f-x1)*(y_f-y1)/((x2-x1)*(y2-y1));  // Blue
807             imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 1] = imgData_s[(y1 * imgstep_s)
                  + (x1 * channels_s) + 1]*(x2-x_f)*(y2-y_f)/((x2-x1)*(y2-y1))
808                                         +imgData_s[(y1 * imgstep_s) + (x2 * channels_s) + 1]*(
                                              x_f-x1)*(y2-y_f)/((x2-x1)*(y2-y1))
```

```
809                                     +imgData_s[(y2 * imgstep_s) + (x1 * channels_s) + 1]*(
                                            x2-x_f)*(y_f-y1)/((x2-x1)*(y2-y1))
810                                     +imgData_s[(y2 * imgstep_s) + (x2 * channels_s) + 1]*(
                                            x_f-x1)*(y_f-y1)/((x2-x1)*(y2-y1));   // Green
811           imgData_d[(y_w * imgstep_d) + (x_w * channels_d) + 2] = imgData_s[(y1 * imgstep_s)
                  + (x1 * channels_s) + 2]*(x2-x_f)*(y2-y_f)/((x2-x1)*(y2-y1))
812                                     +imgData_s[(y1 * imgstep_s) + (x2 * channels_s) + 2]*(
                                            x_f-x1)*(y2-y_f)/((x2-x1)*(y2-y1))
813                                     +imgData_s[(y2 * imgstep_s) + (x1 * channels_s) + 2]*(
                                            x2-x_f)*(y_f-y1)/((x2-x1)*(y2-y1))
814                                     +imgData_s[(y2 * imgstep_s) + (x2 * channels_s) + 2]*(
                                            x_f-x1)*(y_f-y1)/((x2-x1)*(y2-y1));     // Red
815         }
816       }
817
818     }
819   }
820 }
821
822 void findBoundary(CvMat *homography_I, CvMat *m_image_corner, int* min_x, int* min_y, int*
        max_x, int* max_y)
823 {
824
825   CvMat *m_world_corner = cvCreateMat(3,4,CV_64FC1);
826   CvMat *m_D = cvCreateMat(4,1,CV_64FC2);
827
828   cvMatMul(homography_I,m_image_corner,m_world_corner);
829
830   for (int i=0; i<4; i++)
831   {
832     cvSet2D(m_D, i, 0,cvScalar(cvmGet(m_world_corner,0,i)/cvmGet(m_world_corner,2,i),
833       cvmGet(m_world_corner,1,i)/cvmGet(m_world_corner,2,i)));
834   }
835
836   *min_x = (int) min(min(cvGet2D(m_D, 0, 0).val[0], cvGet2D(m_D, 1, 0).val[0]), min(cvGet2D(
          m_D, 2, 0).val[0], cvGet2D(m_D, 3, 0).val[0]));
837
838   *min_y = (int) min(min(cvGet2D(m_D, 0, 0).val[1], cvGet2D(m_D, 1, 0).val[1]), min(cvGet2D(
          m_D, 2, 0).val[1], cvGet2D(m_D, 3, 0).val[1]));
839
840   *max_x = (int) max(max(cvGet2D(m_D, 0, 0).val[0], cvGet2D(m_D, 1, 0).val[0]), max(cvGet2D(
          m_D, 2, 0).val[0], cvGet2D(m_D, 3, 0).val[0]));
841
842   *max_y = (int) max(max(cvGet2D(m_D, 0, 0).val[1], cvGet2D(m_D, 1, 0).val[1]), max(cvGet2D(
          m_D, 2, 0).val[1], cvGet2D(m_D, 3, 0).val[1]));
843
844 }
845
846
847 IplImage* warping(IplImage* img, CvMat* H)
848 {
849   int min_x;
850   int min_y;
851   int max_x;
852   int max_y;
853
854   CvMat *m_image_corner = cvCreateMat(3,4,CV_64FC1);
855
856   cvmSet(m_image_corner,0,0,0);
857   cvmSet(m_image_corner,1,0,0);
858   cvmSet(m_image_corner,2,0,1);
859
860   cvmSet(m_image_corner,0,1,img->width-1);
861   cvmSet(m_image_corner,1,1,0);
862   cvmSet(m_image_corner,2,1,1);
863
864   cvmSet(m_image_corner,0,2,0);
865   cvmSet(m_image_corner,1,2,img->height-1);
866   cvmSet(m_image_corner,2,2,1);
867
```

```
868     cvmSet(m_image_corner,0,3,img->width-1);
869     cvmSet(m_image_corner,1,3,img->height-1);
870     cvmSet(m_image_corner,2,3,1);
871
872     CvMat *H_I = cvCreateMat(3, 3, CV_64FC1);
873
874     cvInvert(H, H_I);
875
876     findBoundary(H, m_image_corner, &min_x, &min_y, &max_x, &max_y);
877
878     IplImage *dst = cvCreateImage(cvSize(img->width,(max_y-min_y)*img->width/(max_x-min_x)),
879                                   img->depth,img->nChannels);
880
881     double scalingFactor =  (double)(max_x-min_x)/dst->width;
882
883     transform(img, dst, H_I, scalingFactor, min_x, min_y);
884
885     return dst;
886   }
887
888   int _tmain(int argc, _TCHAR* argv[])
889   {
890     int seed = (int) time(NULL);
891     srand(seed);
892
893     IplImage* img = 0, *img1 = 0;
894
895     // load an image
896
897     //img=cvLoadImage("sample_a.jpg",0);
898     //img1=cvLoadImage("sample_b.jpg",0);
899     //double thres = 0.5;
900     //double ratio = 0.85;
901
902     //img = cvLoadImage("booklet1.jpg",0);
903     //img1 = cvLoadImage("booklet2.jpg",0);
904     //double thres = 0.5;
905     //double ratio = 0.95;
906
907     img = cvLoadImage("box1.jpg",0);
908     img1 = cvLoadImage("box2.jpg",0);
909     double thres = 0.5;
910     double ratio = 0.95;
911
912
913     CvMat* r = cvCreateMat(img->height, img->width, CV_64FC1);
914     CvMat* r1 = cvCreateMat(img->height, img->width, CV_64FC1);
915
916     r = HarrisCornerDetector(img);
917     r1 = HarrisCornerDetector(img1);
918
919     FEATURE *f1_head, *f2_head;
920
921     f1_head = getFeature(img, r, 4000000);
922     f2_head = getFeature(img1, r1, 3000000);
923
924     printf("No. of Features in the left image = %d\n",feature_no(f1_head->next));
925     printf("No. of Features in the right image = %d\n",feature_no(f2_head->next));
926
927     feature_matching_NCC(f1_head, f2_head, thres, ratio);
928
929     FEATURE* c[4];
930     CvMat* H;
931     int no_inlier;
932     CvMat* H_final;
933     int largest_no_inlier=-1;
934     FEATURE* inlier_head;
935
936     double disThres = 15;
937
```

```
938   long long int N = LLONG_MAX;
939   int sample_count = 0;
940   double e;
941   int total=0;
942   double p = 0.99;
943   FEATURE* corr_head;
944
945   while (N>sample_count)
946   {
947     corr_head = computeCorrespondences(f1_head,c);
948
949     H = computeNormalizedDLT(corr_head);
950
951     no_inlier = computeInlierSetNumber(f1_head, H, disThres, &total);
952     if (no_inlier>largest_no_inlier)
953     {
954       largest_no_inlier = no_inlier;
955       H_final = H;
956     }
957
958     e = 1 - (double) no_inlier/total;
959     if (log(1-pow((1-e),4))!=0)
960       N = (long long int) (log(1-p)/log(1-pow((1-e),4)));
961     else
962       N = LLONG_MAX;
963
964     sample_count++;
965
966   }
967
968   printf("total = %d e = %.3lf N = %lld inlier = %d\n",total, e, N, largest_no_inlier);
969
970   inlier_head = computeInlierSet(f1_head, H_final, disThres);
971
972   CvMat* H_inlier;
973
974   H_inlier = computeNormalizedDLT(inlier_head);
975
976   IplImage *dst;
977   dst = warping(img, H_inlier);
978
979   IplImage *img_h = cvCreateImage(cvSize(img->width*2, img->height), img->depth, 3);
980   cvSetImageROI( img_h, cvRect(0, 0, img->width, img->height) );
981   cvCvtColor(img, img_h, CV_GRAY2BGR);
982
983   markFeatures(img_h, f1_head);
984
985   for (int i=0; i<4; i++)
986     cvCircle(img_h,cvPoint((int)c[i]->pt[0], (int)c[i]->pt[1]),5,cvScalar(255, 0, 0),2);
987   cvResetImageROI(img_h);
988
989   cvSetImageROI( img_h, cvRect(img->width, 0, img->width, img->height) );
990   cvCvtColor(img1, img_h, CV_GRAY2BGR);
991   markFeatures(img_h, f2_head);
992   cvResetImageROI(img_h);
993
994   drawMatching(img_h, inlier_head);
995
996   cvSaveImage("result-corr.jpg",img_h);
997   cvSaveImage("result-transform.jpg",dst);
998
999   // create a window
1000  cvNamedWindow("Result");
1001  cvNamedWindow("Result transform");
1002
1003  cvNamedWindow("Left");
1004  cvNamedWindow("Right");
1005  cvShowImage("Result", img_h );
1006  cvShowImage("Result transform", dst );
1007
```

```
1008     cvShowImage("Left", img);
1009     cvShowImage("Right",img1);
1010
1011
1012     // wait for a key
1013     cvWaitKey(0);
1014
1015     // release the image
1016     cvReleaseImage(&img );
1017
1018     return 0;
1019   }
```